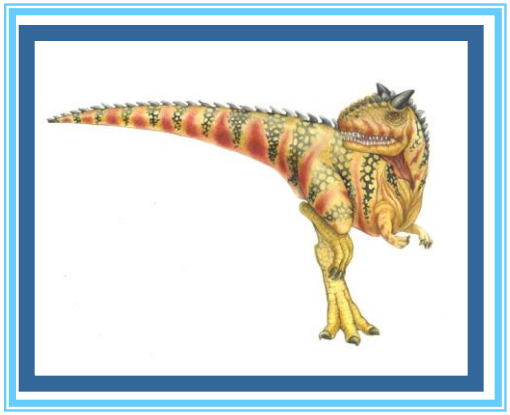
Chapter 3: Process-Concept



Chapter 3: Process-Concept

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

Objectives

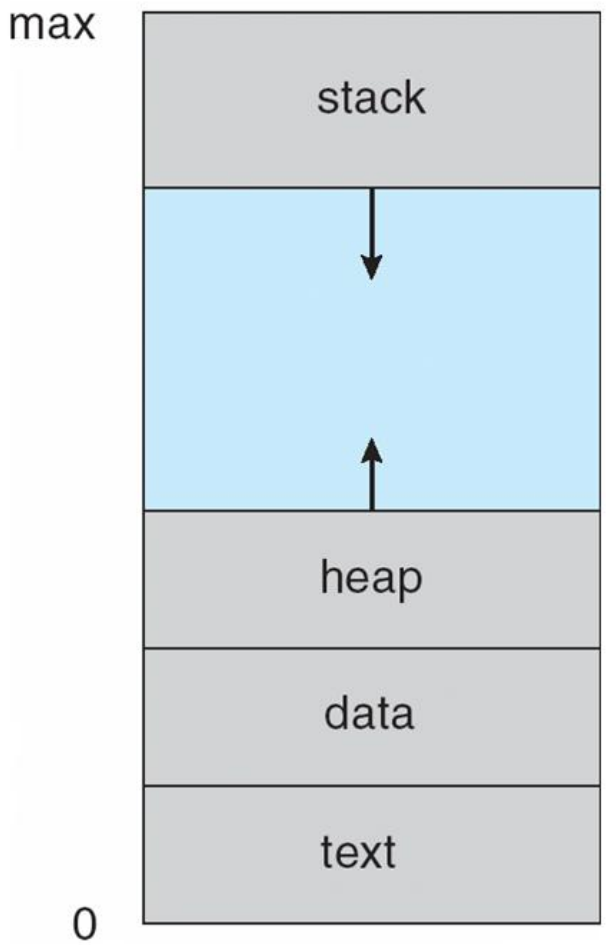
- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes:
  - scheduling,
  - creation and termination,
  - and communication

Process Concept

- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - Text section
  - program counter value
  - Registers contents
  - Stack (temporary data)
  - data section
  - Heap

(See Text Book)

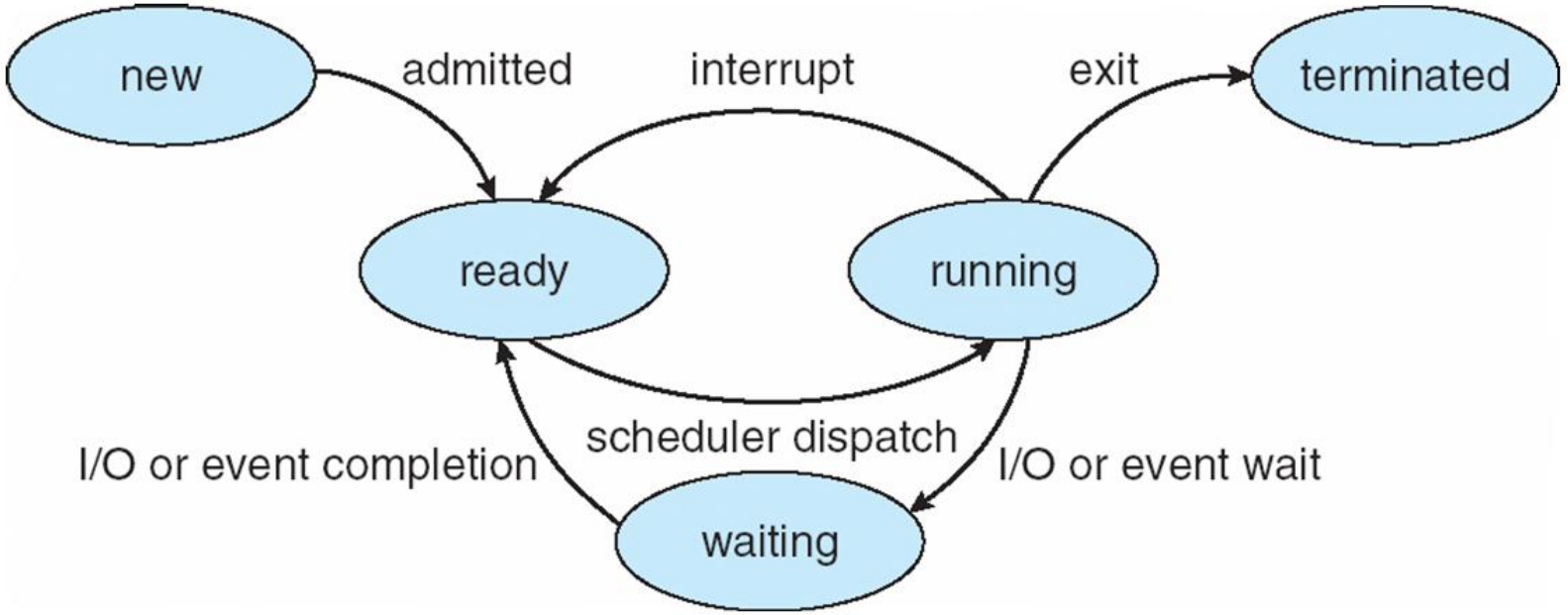
Process in Memory



Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Diagram of Process State



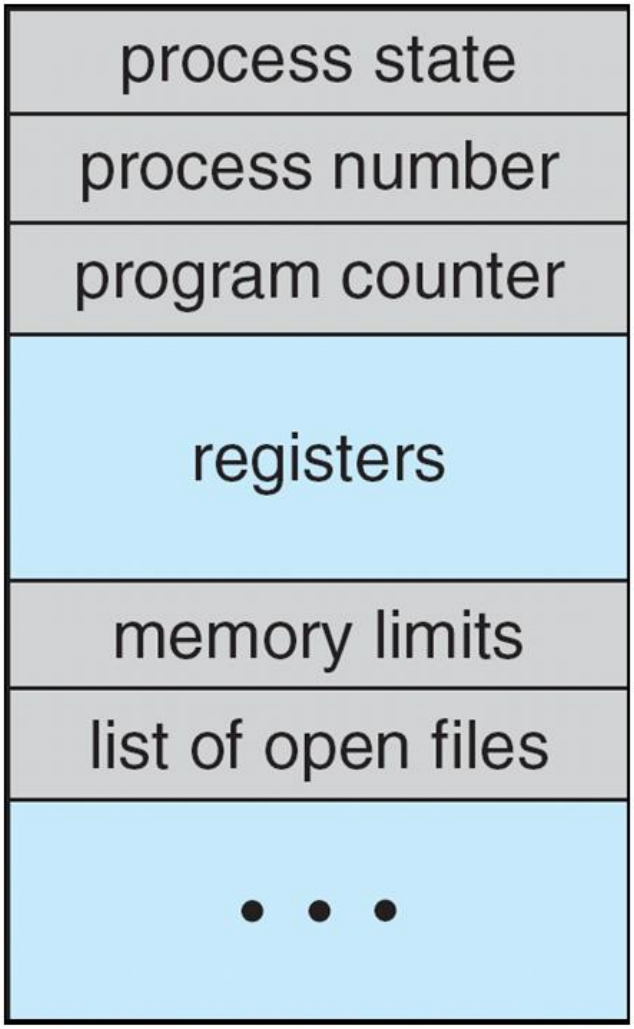
Process Control Block (PCB)

Information associated with each process

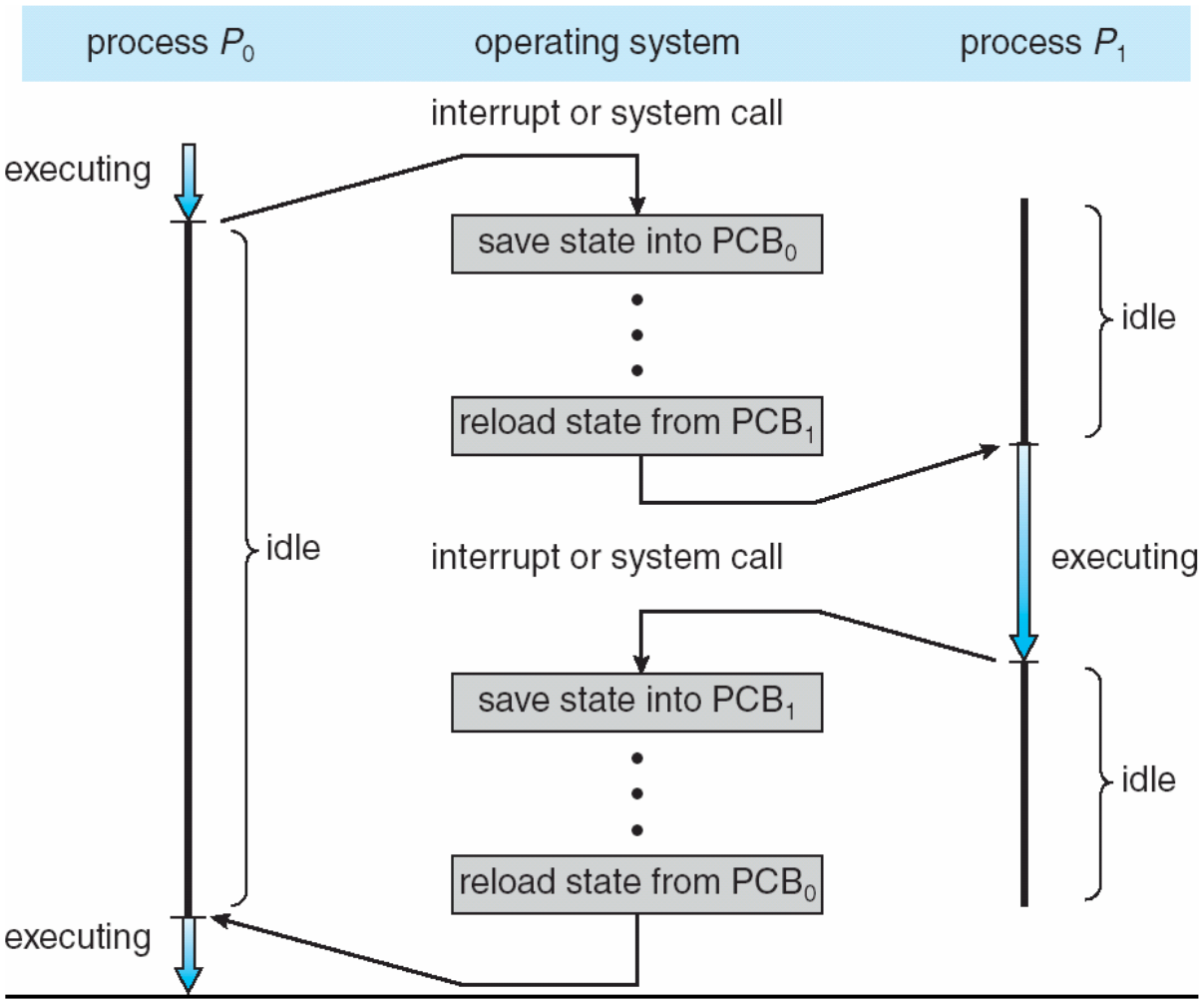
- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

(See Text Book)

Process Control Block (PCB)



CPU Switch From Process to Process



Threads

- **Single thread:** allows the process to perform only one task at one time.
- **Multiple thread:** allows to perform more than one task at a time.
  - The PCB is expanded to include information for each thread.

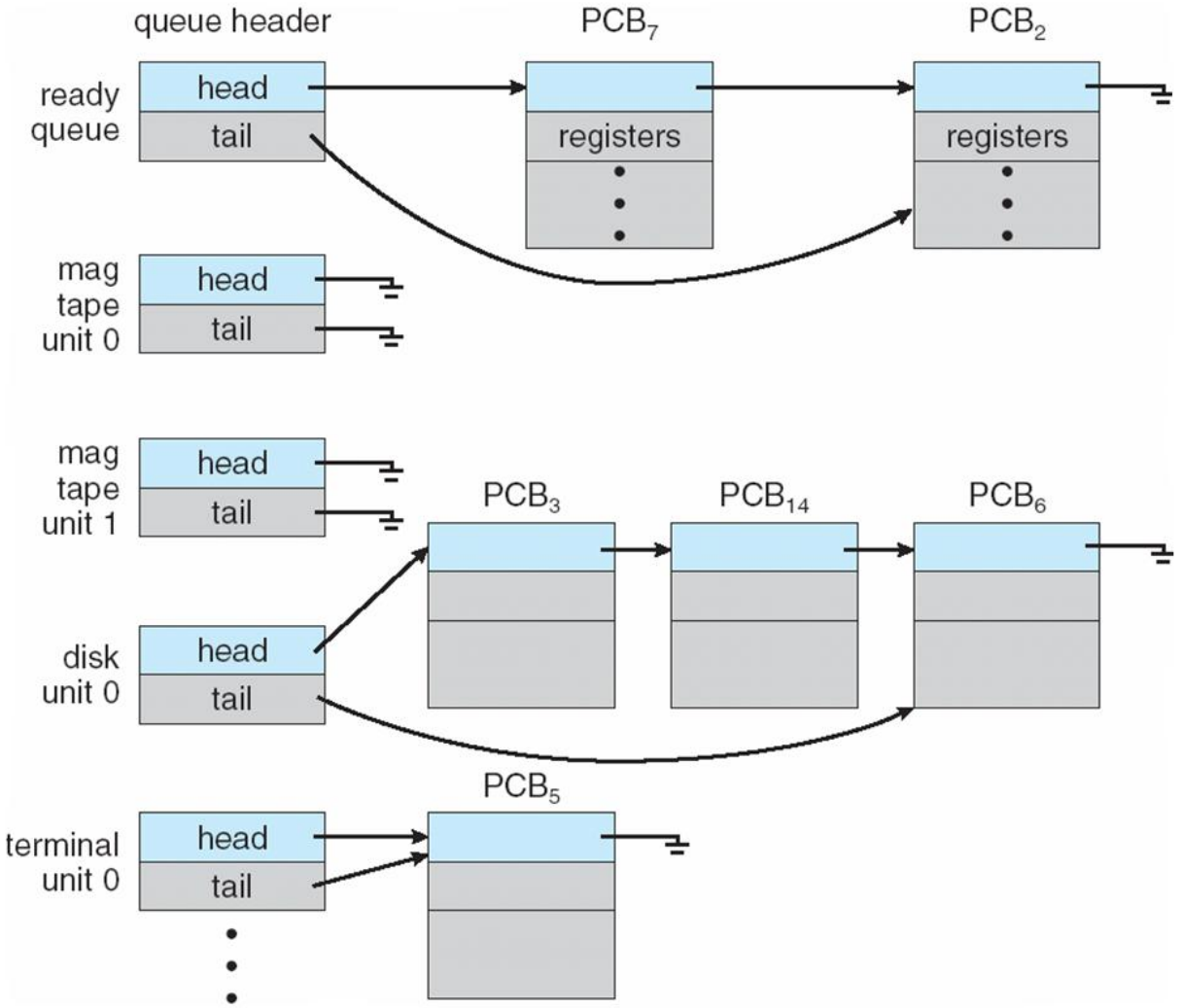
Process Scheduling

- Multiprogramming have some process running at all times to maximize CPU utilization.
- Time sharing switch the CPU among processes so frequently user interact with each program while it is running.
- ***Process scheduler*** selects an available process for program execution on the CPU.

Process Scheduling Queues

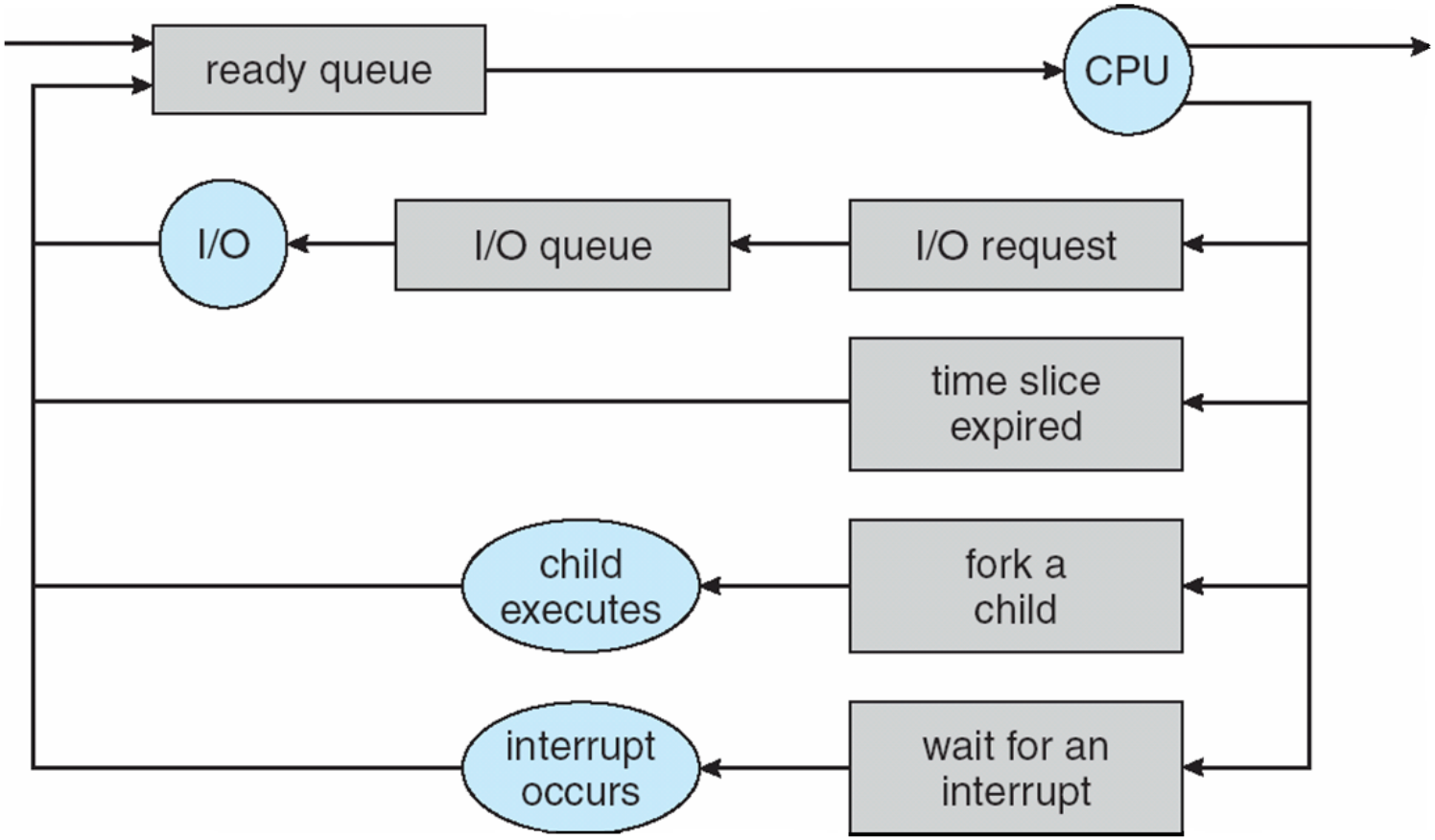
- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

Queuing diagram



(See Text Book)

Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- The long-term scheduler select a good **process mix** of I/O-bound and CPU-bound processes.

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Operation on Processes

- Process Creation
- Process Termination
- Process communication

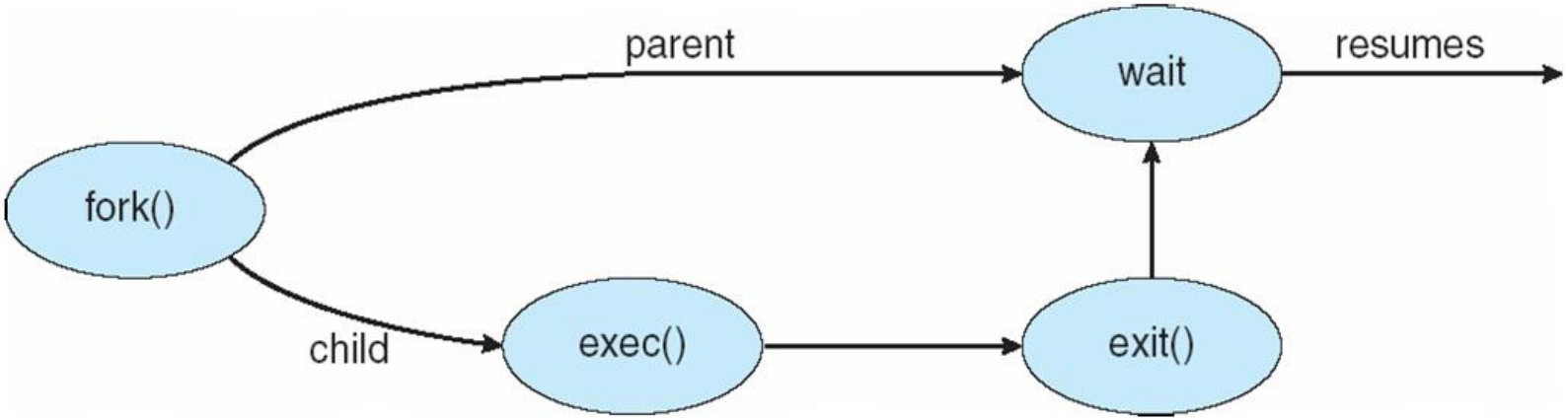
Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via **a process identifier** (**pid**)
- *Resource sharing*
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- *Execution*
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont)

- *Address space*
  - Child duplicate of parent
  - Child has a program loaded into it
- *UNIX Examples*
  - **fork()** system call creates new process
  - **exec()** system call used after a **fork()** to replace the process's memory space with a new program

Process Creation



C Program Forking Separate Process

```
int main()
{
pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent process */
                /* parent will wait for the child to complete */
                wait (NULL);
                printf ("Child Complete");
                exit(0);
        }
}
```

Process Termination

- Process executes last statement and asks the operating system to delete it by using **exit**() system call
  - Output data from child to parent via **wait**() system call
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

Interprocess Communication

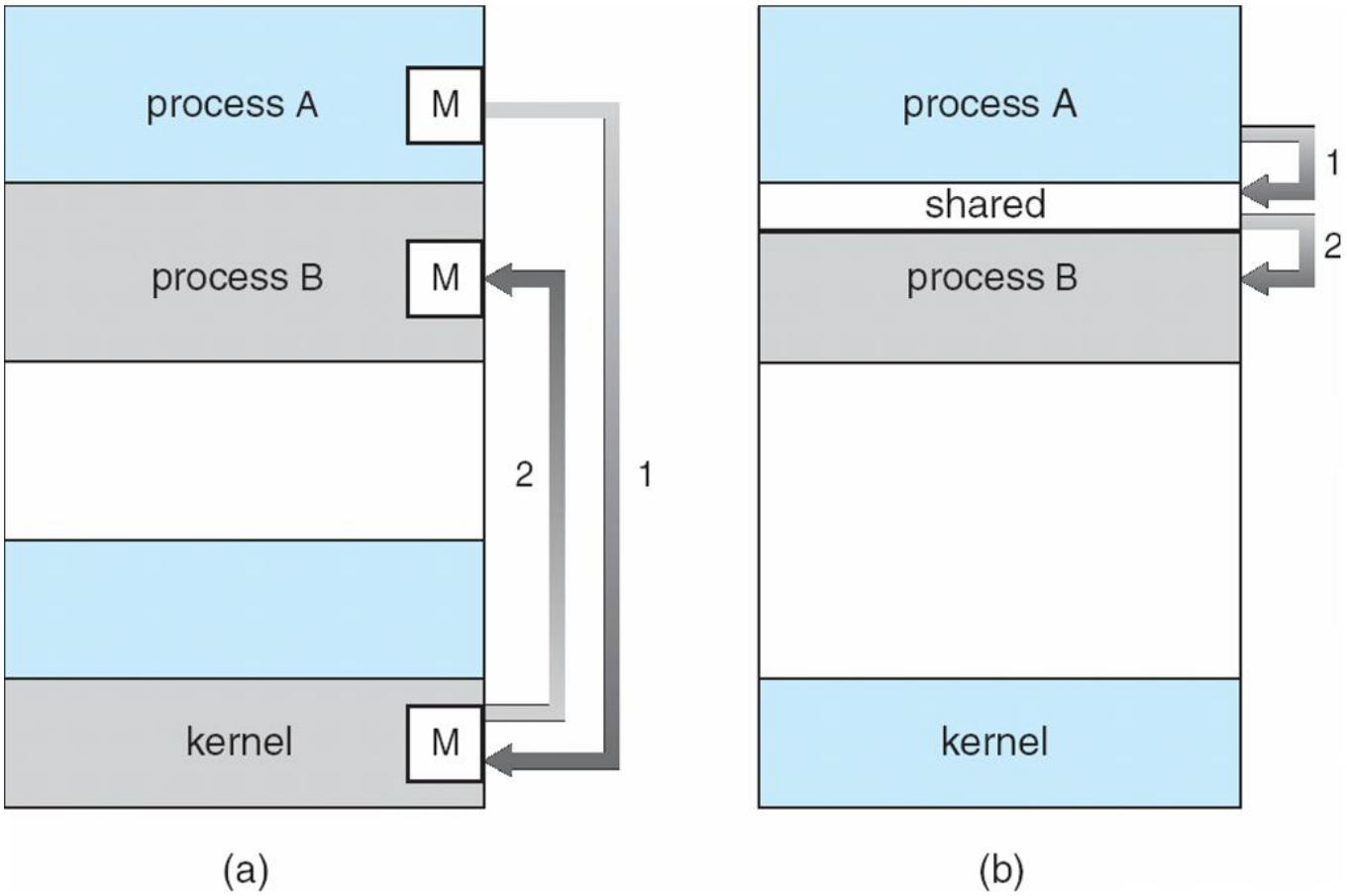
- Processes within a system may be ***independent*** or ***cooperating***
  - Independent processes
    - » cannot affect or be affected by the execution of another process.
  - Cooperating processes
    - » can affect or be affected by the execution of another process
- Reasons (advantages) for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience (e.g. editing, printing, compiling)

Interprocess Communication

- Cooperating processes need **Interprocess Communication** (**IPC**)

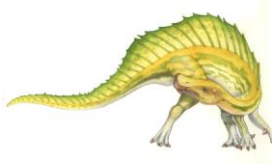
- **Two models of Interprocess Communication (IPC):**
  a. Message passing
  b. Shared memory

IPC Communications Models



Producer-Consumer Problem

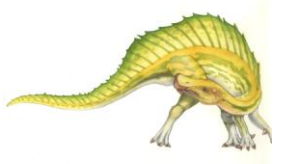
- The common Paradigm for cooperating processes;
  - ***producer*** process produces information that is consumed by a ***consumer*** process.
- To allow producer and consumer processes to run concurrently, We need buffer of items that can be filled by producer and emptied by consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- Producer and Consumer must synchronized, so that the consumer does not try to consume an item that has not yet been produced.
- Two types of buffers can be used:
  - ***Unbounded-buffer*** places no practical limit on the size of the buffer. Consumer may wait, producer never waits.
  - ***Bounded-buffer*** assumes that there is a fixed buffer size. Consumer waits for new item, producer waits if buffer is full.



Bounded-Buffer – Shared-Memory Solution

- Shared data (the following variables reside in a region of memory shared by producer and consumer)

```
#define BUFFER_SIZE 10

typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE]; /* shared buffer = circular array */

int in = 0;   /* points to the next free position in the buffer */

int out = 0; /* points to the first full position in the buffer */

 /* Buffer empty if in == out */

 /* Buffer full  if (in+1) mod BUFFER_SIZE == out */
```



Bounded-Buffer – Producer

```
item    nextProduced
while (true) {
   /* Produce an item in nextProduced */
   while (( (in + 1) % BUFFER SIZE )  == out)
      ;   /* do nothing -- no free buffers */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER SIZE;
}
```

Bounded Buffer – Consumer

```
item    nextConsumed
while (true) {
        while (in == out)
              ; // do nothing -- nothing to consume

        // remove an item from the buffer
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;
    /* consume the item in nextConsumed */
    }
```

Interprocess Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

End of Chapter 3